Methodology

This section describes the hardware (GPU) platform, the operator system, the programming language and the benchmarks.

A. Hardware Platform

To rigorously assess the adaptability and efficiency of our experimental setup across diverse computational ecosystems, we selected three distinctly configured hardware platforms:

1. **MacBook Air (2022)**: Representing contemporary consumer-grade devices, this laptop is fortified with the Apple M2 chip, a cornerstone of the ARM architecture. The chip boasts an 8-core CPU bifurcated into 4 high-performance cores ensuring peak computational prowess, juxtaposed with 4 energy-efficient cores designed for optimised power conservation. Additionally, the system is complemented by a 10-core GPU and an expansive 24GB of unified memory, facilitating seamless multitasking and graphically intensive tasks. Such devices provide insights into everyday user experiences and the challenges posed by energy-consumption optimisation.[1]
2. **Google Cloud Custom Server**: Epitomising high-end cloud computing infrastructures, this server houses 24 cores of the AMD EPYC 7642 CPU, each core operating at a robust Base Clock of 2.3 GHz[2]. As an adjunct to this CPU, the server integrates the formidable RTX 3090 GPU, armed with 10,496 NVIDIA CUDA cores, which is a testament to its massively parallel processing capability. This ensemble has 24GB of memory, facilitating vast data handling and computation[3].
3. **Dragonboard 820c Development Board**: Catering to the burgeoning world of embedded systems, this board rests on the Qualcomm® Snapdragon™ 820E platform. Intrinsically, it comprises a custom 64-bit Qualcomm® Kryo™ quad-core CPU, adhering to the rigorous standards set by the 96Boards Consumer Edition Extended Version specification under Linaro's aegis. Of significant note is its Qualcomm® Adreno™ 530 GPU, a powerhouse in embedded graphics[4]. Coupled with a comprehensive array of interfaces, this board emerges as an archetype for intricate embedded computing applications, extending our analysis to mobile devices. This development board is also the emphasis of this study.

We will implement benchmark tests on three machines to compare GPU architectures across various scenarios and environments.

B. Operator System

Uxn is a virtual stack machine. This one-page computer, programmable in Uxntal, was designed with an implementation-first mindset and focused on creating portable graphical tools and games. Uxn utilises two stacks, allowing it to tunnel through 127 subroutines and find its way back. It can interface with up to 16 devices or peripherals, such as screens, controllers, or other Uxns. Each instance has its stack and device memory but shares 64kb of memory with the others[7].

Uxn is a compact system with notable features and applicability. Firstly, the small footprint of Uxn positions it as an ideal platform for exploring the potential and efficacy of GPU programming on embedded devices. Moreover, given that Uxn is implemented in C, it offers convenience for researchers. We can effortlessly modify its core code, enabling the system to run concurrently on both CPUs and GPUs without requiring a complete rewrite.

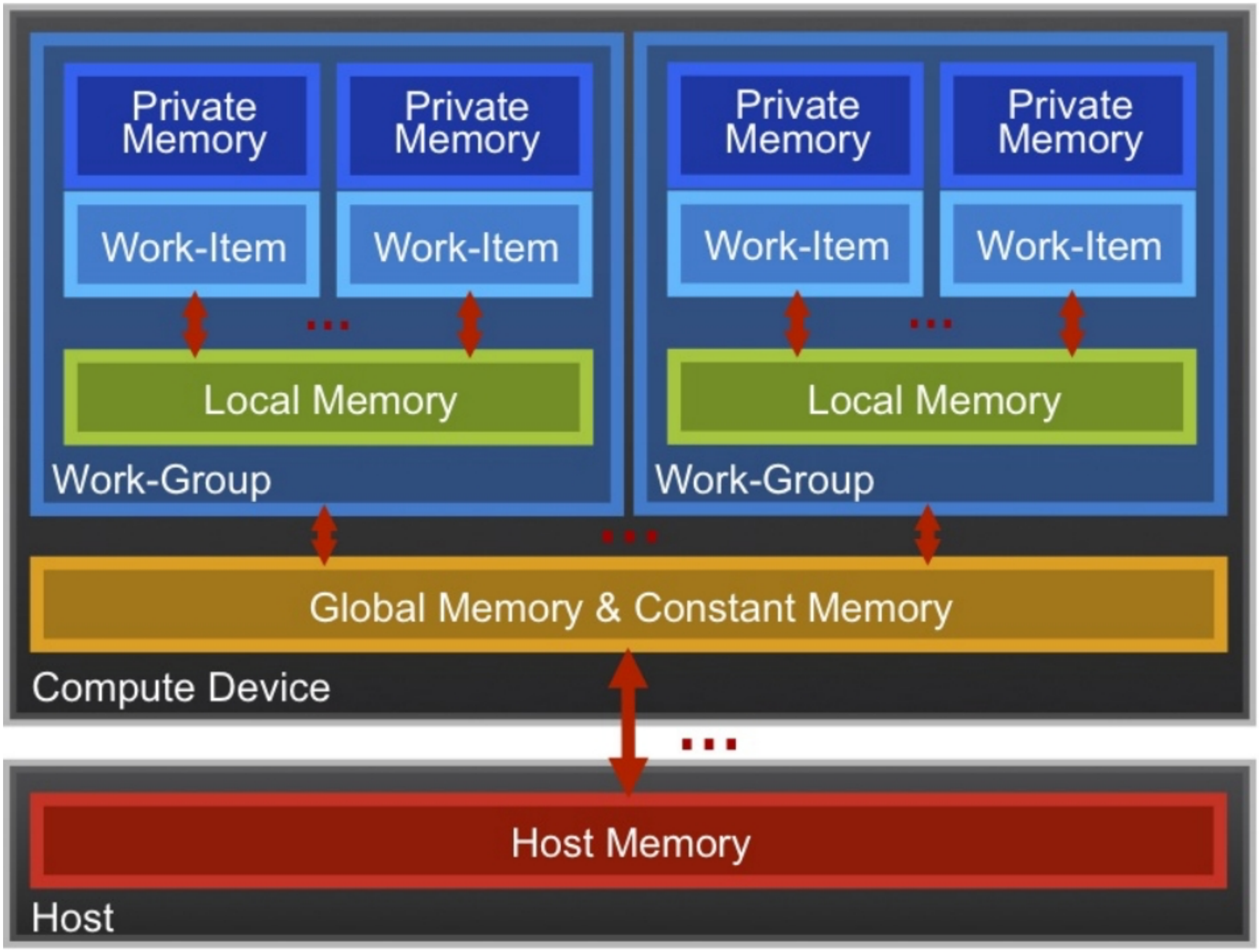
What's more noteworthy is the flexibility Uxn affords to developers. They can write ROM using assembly or Uxn-specific Uxntal language, eliminating reliance on higher-level programming languages. This characteristic ensures a more direct, low-level hardware interface, optimising performance and catering to specific programming requirements.

The Uxn system offers two available emulators: Uxncli (a non-graphical, command-line version) and Uxnemu (with a graphical interface), catering to various usage scenarios. The system's simple yet robust graphical interface augments the possibility of its deployment on mobile terminals, rendering our exploration of its applications on embedded GPUs meaningful.

However, while Uxn boasts several advantages, it also has its constraints. For instance, Uxn only offers 64kb of memory, which limits its utility. Therefore, this paper will refrain from discussing its use in simulating most modern computational activities on embedded GPUs for the time being.

C. Programming Languages

OpenCL (Open Computing Language) [5] is a standard for programming heterogeneous multiprocessor platforms. The standard defines a C language API for invoking “kernels” (functions describing parallel execution on the device) and a C-based language called OpenCL C for defining the kernels. This place will add the picture of memory models[6].



The memory model of OpenCL mandates memory consistency among work items within a work group but not between different work groups. As such, there's no need to be concerned about maintaining memory consistency across computing devices[8], allowing for the initiation of separate work groups on different devices, such as CPUs and GPUs. Consequently, we can manage necessary input-output operations in the CPU while accelerating the computational process in the GPU.

OpenCL also supports writing programs across heterogeneous platforms composed of CPUs, GPUs, and other processing units. In our future endeavours, we mentioned that our ultimate objective is not limited to GPUs but extends to FPGA platforms.

D. Choice of Benchmarks

GPUs allocate fewer resources to hiding memory latency and instead emphasise computational resources. GPGPU can now run programs with a high degree of data parallelism. Therefore, applications with a high proportion of memory access can extract higher performance benefits from the GPU. However, for programs that are parallelised across both CPUs and GPUs, we must consider the overhead of memory transfers between the CPU and GPU. When the frequency and volume of these memory transfers reach a certain threshold over time, we need to evaluate whether our approach can effectively improve program efficiency and leverage the GPU's full performance capabilities.

Hence, the primary consideration for our benchmark selection is to ensure that some benchmarks are highly dedicated to data computation. In contrast, others focus more on data communication between the GPU and CPU. Moreover, we aim to cover a broad spectrum of Uxn application scenarios. As mentioned in the B.Operator System, the Uxn system is dedicated to developing graphic tools and games, so we have added test scenarios related to graphical interfaces.

We have selected our benchmarks based on the previous evaluation work by Wim Vanderbauwhede[9]. We've chosen stencil.tal, fib.tal, and primes.tal for command window assessment, while bunnymark.tal was selected for graphical interface evaluation.

For the command-line interface version of the Uxn system (Uxncli):

1. First and foremost, we opted for **the total time of memory exchange between the CPU and GPU** within the system as our benchmark. Compared to the CPU-exclusive system (the original Uxn system), the memory exchange between the CPU and GPU, an added component in the new system, will inevitably influence the overall system runtime. Hence, we selected it as our initial benchmark to gauge the new system's performance.
2. Secondly, we designated **the system's total runtime** as our benchmark. By contrasting the system's entire runtime, we aim to ascertain whether the CPU/GPU parallel system offers a performance enhancement relative to the CPU-exclusive system, and evaluate the potential for further optimisation.
3. Thirdly, we chose **the runtime within the GPU** as our benchmark. We seek to measure whether the GPU brings about an uplift in computational efficiency for the entire Uxn system, to what extent this uplift occurs, and whether modifying instructions in the Uxn system can enhance the computational efficiency within the GPU.

For the graphical user interface (GUI) version of the Uxn system (Uxnemu):

We selected **the frame rate** of the rendered graphical interface as our benchmark. In reality, within most Uxn system applications, users predominantly interact with the graphical interface and emphasise the user experience it offers. Therefore, given the premise of ensuring system accuracy, a higher frame rate can enhance user experience. As a result, we opted for the graphical frame rate as a pivotal benchmark for evaluating the Uxn GUI system.

[1] Apple. (2022, June). Apple unveils M2 with breakthrough performance and capabilities. Apple Newsroom. https://www.apple.com/newsroom/2022/06/apple-unveils-m2-with-breakthrough-performance-and-capabilities/

[2] AMD. (n.d.). AMD EPYC 7642. Retrieved from https://www.amd.com/en/products/cpu/amd-epyc-7642

[3] NVIDIA. (n.d.). GeForce RTX 3090 and RTX 3090 Ti Graphics Cards. Retrieved from https://www.nvidia.com/en-gb/geforce/graphics-cards/30-series/rtx-3090-3090ti/

[4] 96Boards. (2021). Dragonboard 820c. Retrieved from https://www.96boards.org/product/dragonboard820c/

[5] OpenCL Specification v1.0r48, Khronos Group, Oct. 2009. [Online].

Available: http://www.khronos.org/registry/cl/

[6] Jäskeläinen, P. O., de La Lama, C. S., Huerta, P., & Takala, J. H. (2010). OpenCL-based design methodology for application-specific processors. In *2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*. https://doi.org/10.1109/ICSAMOS.2010.5642061

[7] Anon. (n.d.). Uxn. XXIIVV. Retrieved from https://wiki.xxiivv.com/site/uxn.html

[8] Singh, A. K., Prakash, A., Reddy Basireddy, K., Merrett, G. V., & Al-Hashimi, B. M. (2017). Energy-Efficient Runtime Mapping and Thread Partitioning of Concurrent OpenCL Applications on CPU-GPU MPSoCs. *ACM Transactions on Embedded Computing Systems*, 16(5s), 1-22. https://doi.org/10.1145/3126548

[9] Wim Vanderbauwhede. (2022, October 15). *Compiling stack-based assembly to C*. Wim Vanderbauwhede. https://limited.systems/articles/uxntal-to-C/